Secure Embedded Systems

Michael Vai, David Whelihan, Ben Nahill, Dan Utin, Sean O'Melia, and Roger Khazan

Abstract

Department of Defense (DoD) systems are increasingly the targets of deliberate and sophisticated attacks. In order to assure mission success, military systems must be entrusted to perform its intended functions, prevent attacks, and even operate with resilience under attack. DoD has thus directed that cyber security must be integrated into system life cycles [1], the practice of securing a system after it has been designed is no longer acceptable. However, the co-design of security with functionality has to overcome a major challenge; rarely can the security requirements be accurately identified when the design begins.

This paper gives an overview on Lincoln Laboratory's co-design methodology for secure embedded systems, which consists of an architecture that decouples secure and functional design aspects and a few enabling technologies. This architecture uses cryptography to ensure the confidentiality and integrity of an embedded system being designed. The development of a hypothetical secure embedded system for an unmanned aerial system (UAS) is used to illustrate our co-design methodology. We will also briefly describe our ongoing effort in adding resiliency to a secure embedded system so that it can continue to function under attack for enhanced availability.

Introduction

Lincoln Laboratory is at the research and development (R&D) forefront of leading edge signal processing solutions for challenging critical missions. Many of Lincoln Laboratory's prototype embedded systems must be designed with security in mind, so that they can be quickly brought into compliance with DoD's relevant requirements to support field tests and technology transfers. DoD has now directed that cyber security must be co-developed with mission capabilities and not be treated as an "after-thought," since any after-the-fact system changes could be prohibitively expensive or even impossible [1]. The co-design of embedded system functionality with its security is difficult as rarely can the security requirements be accurately identified when the design process starts. Also, embedded system engineers tend to focus on well-understood functional capabilities rather than obscure security requirements.

The security requirements of an embedded system are determined according to its concept of operations (CONOPS). Security aims at preventing attacks so that a system can be entrusted to perform in support of a successful mission. For critical mission functions, we may need to go a step further and provide resiliency that enables the system to continue functioning, albeit with possibly degraded capabilities, when security fails.

Distribution A: Public Release. This work was sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002 Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Governme nt.

The goals of securing a system can be summarized as the assuring of confidentiality, integrity, and availability, which is often referred to as the CIA triad. We define the CIA triad of an embedded system as follows:

- Confidentiality assures that its critical information, such as application code and surveillance data, will not be disclosed to unauthorized entities.
- Integrity assures that the system operation cannot be altered.
- Availability assures that mission objectives cannot be disrupted.

Security must be provided with minimal impacts on the system's size, weight, power, and cost (SWaP-C) and development schedule. In addition, the design must consider usability as valid secure systems must also be practical and usable.

The objective of this paper is to provide an overview of Lincoln Laboratory's secure embedded system development methodology and its enabling technologies. We use the development of a hypothetical secure unmanned aerial system (UAS) embedded system as an example to illustrate how we use cryptography to ensure confidentiality and integrity. Using this example, we demonstrate the identification of potential attacks by considering its CONOPS, countermeasures to these attacks, and discuss the design and implementation of a cryptography-based security architecture. We will also overview ongoing work to extend the methodology and provide the resiliency required for availability, which is not provided by cryptography itself.

A comprehensive security technology survey is beyond the scope of this paper. Instead, we introduce a number of relevant security technologies that Lincoln Laboratory has been developing to address existing technology gaps in the security of embedded systems.

Challenges in Securing Embedded Systems

An embedded system is a computer system designed for a dedicated function. This is in contrast to a general-purpose computer system (e.g., a desktop computer). A major objective of creating an embedded system for a specific task is to optimize it for better performance in terms of smaller form factor, lower power consumption, higher throughput, etc. As such, an embedded system will provide very little, if any, SWaP allowance for security. Security thus must not impose excessive overheads on the system they are protecting.

Every year DoD acquires and operates numerous embedded systems, ranging from intelligence, surveillance, and reconnaissance (ISR) sensors to electronic warfare/electronic signal intelligence (EW/ELINT) applications [1]. Depending on their objective mission CONOPS, embedded systems have different security requirements. For example, a system planned for contiguous United States (CONUS) deployment could have lower security requirements than a system for forward operating base (FOB) operation, since the latter are deployed to support tactical operations. Any methodology for securing embedded systems should be customizable to meet CONOPS needs.

It is interesting to note that while DoD has some of the most demanding applications, in terms of throughput and SWaP-C, the time that DoD drives technology development has long passed. Instead, DoD now strives to leverage commercial technologies, which are driven by computing, communications, and gaming industries. The result is that the critical technology in a system is often its software and/or firmware, which define the system functionality, rather than the processor hardware itself. Security technologies must be compatible with embedded systems using COTS (commercial-off-the-shelf) processor hardware platforms.

As military electronic systems continue to increase in sophistication and capability, the cost and development time of these systems also grow. The use of open systems architecture (OSA) can improve the development and lifecycle efficiency of asset procurements. DoD has thus directed that all DoD components and agencies use OSA for acquisition [3]. OSA uses industry consensus, non-proprietary, system architectural standards so that variable payloads can be shared among variable platforms. Competition for technology refresh and enterprise-level acquisition are thus promoted. However, adding security to OSA, if not well-thought-out, could interfere with its openness. As most current security approaches are ad-hoc, proprietary, and expensive, they are incompatible with OSA principles, especially when each component individually implements and manages its own security. A system level secure architecture that will seamlessly work with various OSA components is a challenge.

The current trends of how DoD acquires its embedded systems require new thinking on their development for security. We will explain the secure embedded system design methodology that Lincoln Laboratory has developed to address the challenges in securing DoD embedded systems.

Design Process

Figure 1 captures the design process of a secure embedded system with the steps dedicated to security highlighted. CONOPS is developed from the mission objectives and used to derive both functional and security requirements. An initial system design is created, evaluated, and implemented. While functionality and security must be co-developed, it is desirable to decouple their implementations so that the interference of security to the test and evaluation of functionality is minimized. Note that several design iterations may be required before the mission objectives are met.

We use the design of a hypothetical small unmanned aerial system (UAS) for a video surveillance application to illustrate the secure embedded system design process. The CONOPS of this example UAS application is as follows. At startup, the UAS is provisioned on the ground by loading its long term credentials for identification and authentication purposes. Mission specific software and firmware, which are considered to be critical program information (CPI) as they contain information on destinations, targets, search and track algorithms, etc., are loaded into their respective memories. The system is then booted up and prepared for execution.

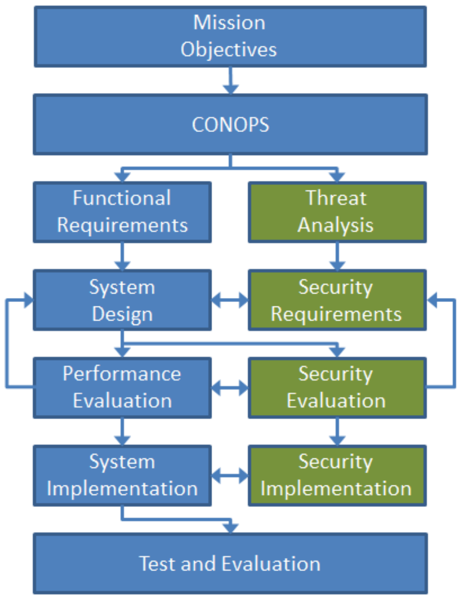


Figure 1: Secure embedded system design process.

Figure 2 illustrates the embedded system in its execution phase. Under the command and control of a ground control station (GCS), the UAS takes off, flies to destination, and begins to collect video data. Target information obtained by processing video data is encrypted and broadcasted to authorized ground stations using the radio. Raw video data are also saved in the storage for further processing after landed. At shutdown, both raw and processed video data are considered to be sensitive and must have been saved securely. Finally, when the system is off, any persistent state, such as the long-term credentials, must be protected.

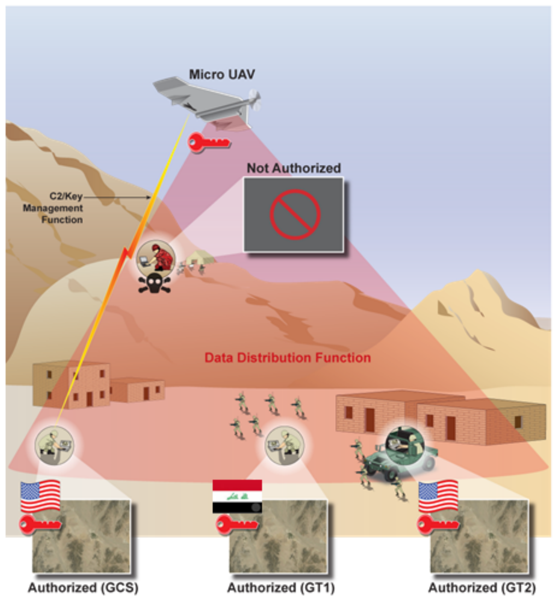


Figure 2: Example UAS application in its execution phase.

Figure 3 shows a high-level architecture initially designed for the functionality of this example UAS embedded system. It consists of a central processing unit (CPU) and a field programmable gate array (FPGA) interconnected with an Ethernet network. The CPU will be provided with its basic input/output system (BIOS), operating system (OS), and mission specific application code in memory. The FPGA has its configuration stored in a firmware memory. Besides a video camera payload, the system also has memory, storage, and radio, which are accessible by the CPU and/or FPGA through Ethernet. The CPU handles command and control received through the radio. The video signal is first processed by the FPGA (e.g., for target detection and identification) and then passed to the CPU for information extraction (e.g., target tracking).

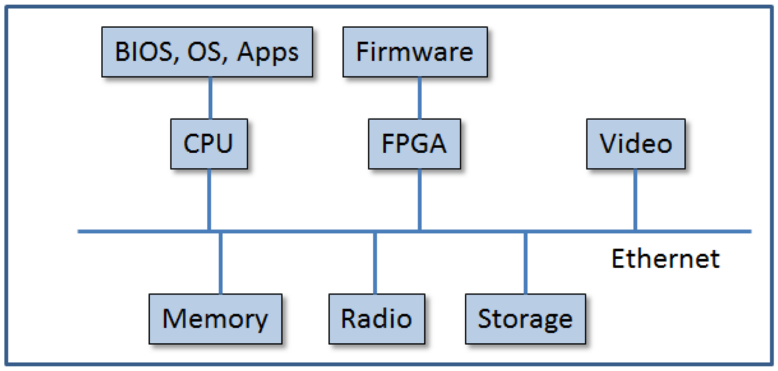


Figure 3: Example UAS embedded system functional architecture.

High performance embedded system design has been the subject of many books (e.g., [2]). The functionality design of this UAS is thus beyond the scope of the paper and will not be further elaborated. Instead, we describe some general security-related considerations on the selection of processing elements (i.e., the CPU and FPGA). The processing elements must be chosen to securely deliver the UAS functionality requirements. This UAS application involves sophisticated signal processing and requires high throughput (e.g., measured by the number of floating point operations per second) with a stringent SWaP allowance.

In order to support a complicated signal processing algorithm, the CPU will need large memory and storage. The choice of a popular "mainstream" processor will allow the leverage of available software libraries in application development, but it may not have the security features desired for the CONOPS. On the other hand, a "secure" processor with built-in security features may simplify system development, but it may not possess the appropriate processing power or may not support large memory space required for the application. Besides, system openness and upgradability must be considered before choosing a secure processor over a mainstream CPU.

Lincoln Laboratory has developed and demonstrated a secure embedded system architecture that uses a security co-processor (S-COP) to secure a mainstream CPU. We will describe this technology later in this paper.

Many popular FPGAs are built with embedded security features [4]. FPGAs should be selected on their capability to encrypt and authenticate their configuration bit-streams, incorporate security monitors to detect attacks, and erase decryption keys to protect the CPI, in this case its firmware, when attacks are detected. Ideally, the FPGA design flow should allow some level of fault containment and tolerance by using modular redundancy, watchdog alarms, security level segregation, and test logic (e.g., the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture) disabling.

Threat Analysis

Adversaries want to sabotage U.S. missions and/or develop counter-measures. The first step in designing a secure system is to analyze the potential attacks that the system may be subjected to when it is deployed. Its CONOPS determines not only functional requirements, but also potential attacks of an adversary. The attacks depends on the adversary capability (e.g., a nation-state) and its attack objectives (e.g., to exfiltrate CPI). In the UAS example, we assume that there is a high probability of equipment loss due to the small UAS size and its use in hostile areas. The UAS attack tree example in Figure 4 describes three logical attack surfaces: boot process, system data, and software, and one physical attack surface: physical system, against which adversaries may attack to exfiltrate CPI.

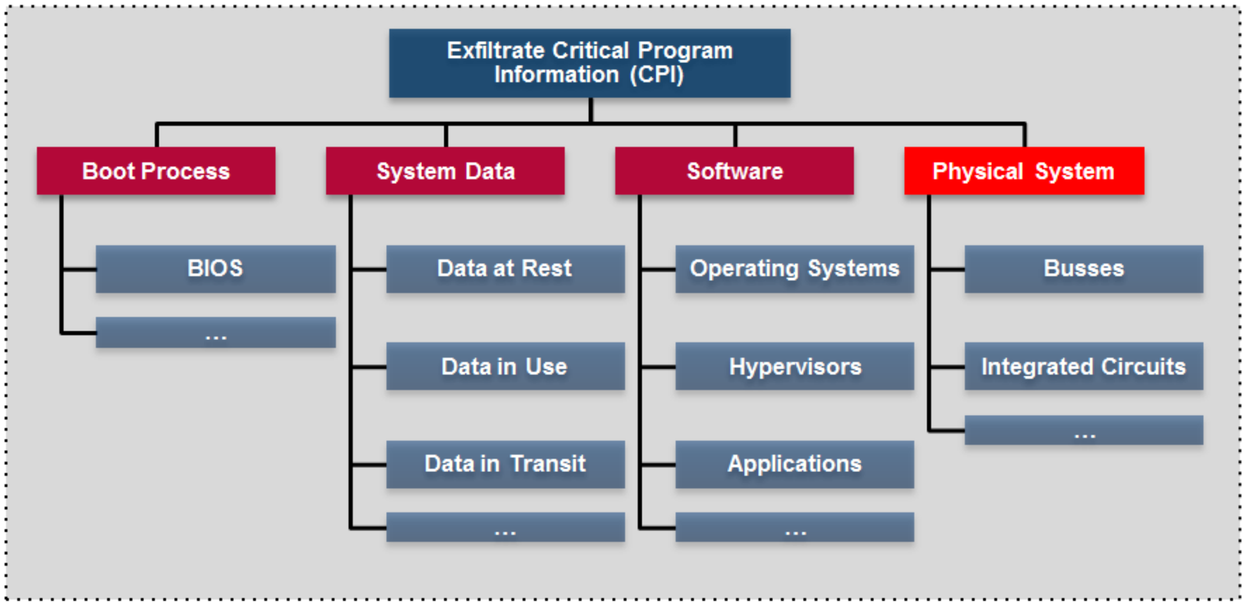


Figure 4: Example UAS attack tree.

A secure system must establish a root of trust when it starts. The CPU boot process is thus the foundation of a secure embedded system and must be protected from attacks on its confidentiality and integrity. Current practice uses a trusted platform module (TPM) to authenticate software components [5]. A TPM offers facilities for the secure generation of cryptographic keys and limitation of their use. It also includes capabilities such as remote attestation, encryption, decryption, and sealed storage. A software application can also use a TPM chip to authenticate hardware devices. Since each TPM chip has a unique and secret key burned in as it is produced, it is also capable of performing platform authentication.

TPM, being a commercial product, made a number of compromises to ensure adoptability by cost-averse vendors and privacy concerns of the general public. The manufactured parts had to be inexpensive and cause as little disruption to the current processing architecture. Additionally, privacy concerns forced that the use of the device to be an optional and passive part of the processing system operations. These compromises led to a low performance device without adequate physical protection. In the next section, we will introduce Lincoln Laboratory's S-COP security coprocessor equipped with a physical unclonable function (PUF), which was developed to address the TPM's inadequacy in tactical operations.

Alternatively, the latent vulnerability (e.g., bugs) within an authorized software component (e.g., from open source) may be exploited to compromise system operations. Adversary may exploit these vulnerabilities to access critical data or gain control of the platform itself. Even though when only authorized users are allowed to access the system, threats could come from the introduction of untrusted software (e.g., malware) or unwanted functionality through a third party's intellectual property (IP), either through malfeasance or negligence. A secure system must prevent compromised software from giving an attacker unfettered system access. Commercial systems are starting to address these issues. Software developers use separation kernels to establish and isolate multiple partitions and control information flow between these partitions. On the hardware side, for example, Intel has been developing a Secure Guard Extensions (SGX) that enforces separations between threads executing on the same processor [6].

Since the UAS is built with minimal system software for dedicated purposes, the exploitation of software vulnerability may be less likely than a general-purpose computer. The strictly controlled provisioning environment accessible by a very limited number of authorized users also reduces the risk of introducing unverified and/or untrusted software to the UAS.

One should always assume that an adversary will attempt to eavesdrop on the UAS data. For example, as the UAS communicates wirelessly, an adversary could potentially eavesdrop on the communication. Data protection is thus a high priority for a secure processor. We need to protect the confidentiality and integrity of the UAS data existing in three forms: data-in-use, data-at-rest, and data-in-transit. Various hardware and software cryptographic solutions, for example, self-encrypting drives[1] and HAIPE (High Assurance Internet Protocol Encryptor), are available. However, the challenge is that encryption must be fully integrated with the processor for efficient performance and protection. Also, the effectiveness of encryption depends on its key management. We will explain an embedded system security framework that is based on the Lincoln Open Cryptographic Key Management Architecture (LOCKMA) library.

[1] Self-encrypting drives, commonly used in laptop protection, are not adequate for classified information protection.

Physical attack is a particular threat to tactical systems. The UAS needs to consider physical attack since there is a higher probability for adversaries to gain physical access to the device, e.g., by capturing it. One must then assume that the adversary could have an extended period of time to reverse engineer the system. The adversary may want to modify or reverse engineer sensitive system components. Protection against physical attack is also required in foreign military sales (FMS). Like an adversary, a foreign nation may attempt to explore the system that it has purchased, either to leapfrog its own technology, or to gain unauthorized capabilities. The most popular technique to date is to use a protective enclosure to delay unauthorized accesses, which has to deal with the challenge of maintaining standby power for intrusion detection and responses.

Security Metrics

In this section we define a few security metrics for evaluating a system during its design process. One of the challenges in the development of secure embedded systems is the inherent difficulty in quantitatively specifying and measuring security. Based on the CIA triad and usability design principles, we have created three practical security metrics to facilitate the design of a secure embedded system:

Trustworthiness - a qualitative measure of the system's effectiveness in defensing against potential threats relevant to its CONOPS. Based on the current system design and the confidence in the fidelity of that information, one can develop a certain level of trust in its behavior. The fact that a system is equipped with a defense mechanism against a certain threat apparently improves its security and thus trustworthiness. However, while unprotected vulnerabilities reduce security, it is valuable to understand the current system's vulnerabilities; one can apply the protection metric (described next) to improve the design by supporting "added-on" protection technologies to address its vulnerabilities.

Protection - a qualitative measure of the system's capability to support "added-on" protection technologies and address vulnerabilities in a CONOPS. The system security can be expressed as a function of the trustworthiness and protection metrics, as they together gauge its security, vulnerabilities, and protection.

Usability - a qualitative measure of the system's suitability for a particular CONOPS. A system that is highly secure but incapable of delivering the required functionality is nevertheless not a valid design. This metric evaluates a system design by considering its throughput, portability, upgradability, SWaP (size, weight, and power) and other similar parameters.

These security metrics do not support absolute measurements, but provide parameters to guide the design of security into an embedded system as its mission functionality architecture evolves. In addition, multiple system architectures can be qualitatively evaluated and compared to determine relatively how well they provide security. As these metrics are by nature qualitative and subjective, sufficient documentation of justification must be kept for each decision made.

The processing requirements, threats and protection needs of a system do not stay constant over the course of its operation. We thus define four operational phases for a secure embedded system so that it can be evaluated accordingly:

Startup – In this phase, a system is being "booted" into a state suitable for operations. A Trusted Computing Base (TCB), which defines the components being trusted for security, is established.

Execution – The system is in the state of operation and performs functions required by the mission.

Shutdown – In this phase, the system is in the process of turning off.

Off – The system has been powered down.

Secure Embedded System Architecture and Enabling Technologies

As mentioned, the CPI of a commercial-off-the-shelf (COTS) based embedded system is mostly in its software and firmware so encryption is the foundation of its overall security. There are many efficient and iron-clad secure cryptographic primitives, such as the NSA approved Suite B cryptography [7]. These primitives can be implemented with software, firmware, or hardware, and can often be obtained as open-source IPs (intellectual properties). However, simply using "standard" cryptographic primitives cannot guarantee the adequate implementation of security functions. The manner that the cryptographic primitives are assembled and coordinated into the desired application-specific security functions is critical to their effectiveness. Also, encryption effectiveness depends on key management, which includes the generation, distribution, and protection of keys.

Lincoln Laboratory has developed a solution to address these challenges. Lincoln Laboratory Open Cryptographic Key Management Architecture (LOCKMA) is a highly portable, modular, open software library of key management and cryptographic operations that are suitable for embedded uses. Designed to secure a wide range of missions, it provides user, identity, and key management and supports for hardware and software cryptographic primitive kernels. LOCKMA's frontend application programming interface (API) provides application developers with a simple and intuitive access to LOCKMA's core functionality. Figure 5 summarizes LOCKMA's interfaces to high level security functions and low level crypto primitives. To use LOCKMA, developers are not required to have advanced knowledge of the cryptography or key management algorithms implemented by LOCKMA's core modules. These modules handle the processing of key management messages. They make extensive use of cryptographic primitives available in several commercial and Open Source libraries.

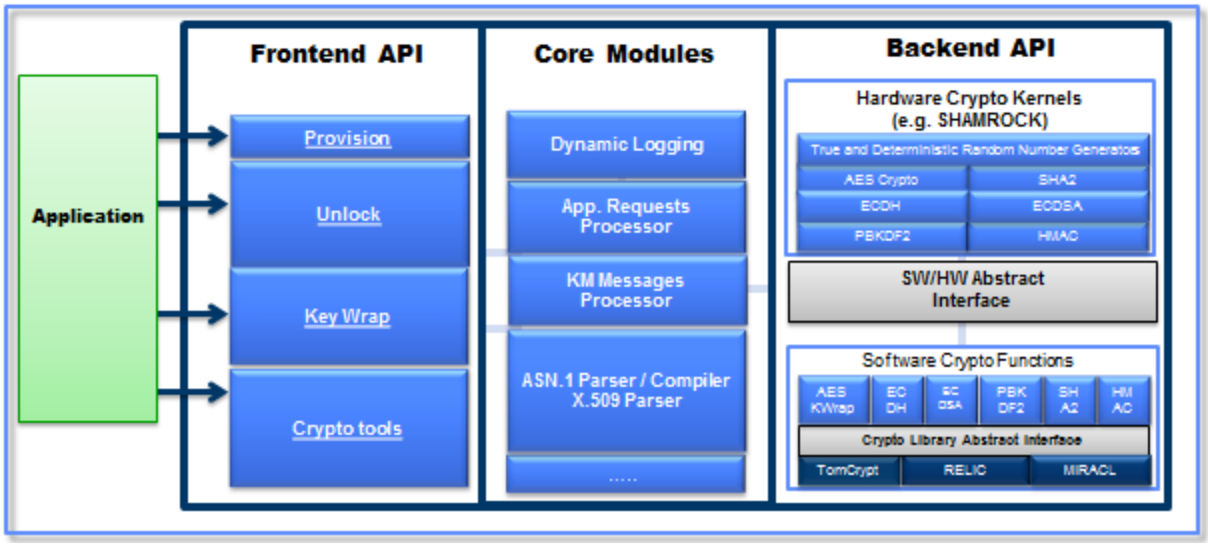


Figure 5: LOCKMA provides application interface, high level security functions, and low level cryptographic kernels.

Lincoln Laboratory has implemented LOCKMA in a security coprocessor (S-COP), which implements cryptographic primitives in hardware. The added benefits of hardware implementation include much faster computation times, lower power use, and hardware separation and protection of sensitive keys from non-sensitive data and code. We have demonstrated the use of S-COP to secure an embedded system.

Figure 6 shows the UAS embedded system architecture in which the CPU secured with an S-COP and a physical unclonable function (PUF). The S-COP employs dynamic key management and accelerated Suite B cryptography for the measurement and verification steps necessary to securely boot the CPU. The PUF provides an inviolable root-of-trust, from which a unique cryptographic key is derived.

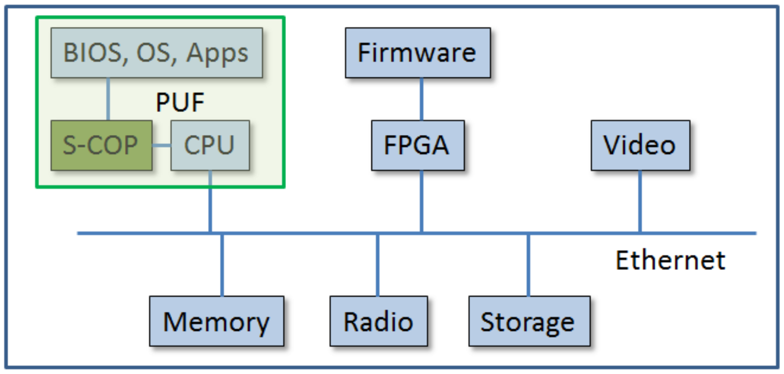


Figure 6: An S-COP is used along with a PUF to secure a COTS CPU.

Lincoln Laboratory has adopted an optical PUF, which can be implemented on a fully fabricated printed circuit board (PCB). As illustrated in Figure 7, the PUF is constructed by adding one or more light emitting diodes (LEDs) and an imager to the PCB, which is then coated with a thin polymer planar waveguide. Upon power-up, the S-COP derives a unique value from the imager, which receives light emitted by the LEDs and travelled through the waveguide. This value is then used for identification and key derivation. Manufacturing variations ensure a unique identification value for each board. Invasive attempts to learn about the PUF value (e.g., for cloning purposes), even when the PCB is unpowered, will disturb and damage the coating and irreversibly destroy the PUF value. Cloning and other unauthorized actions are thus prevented.

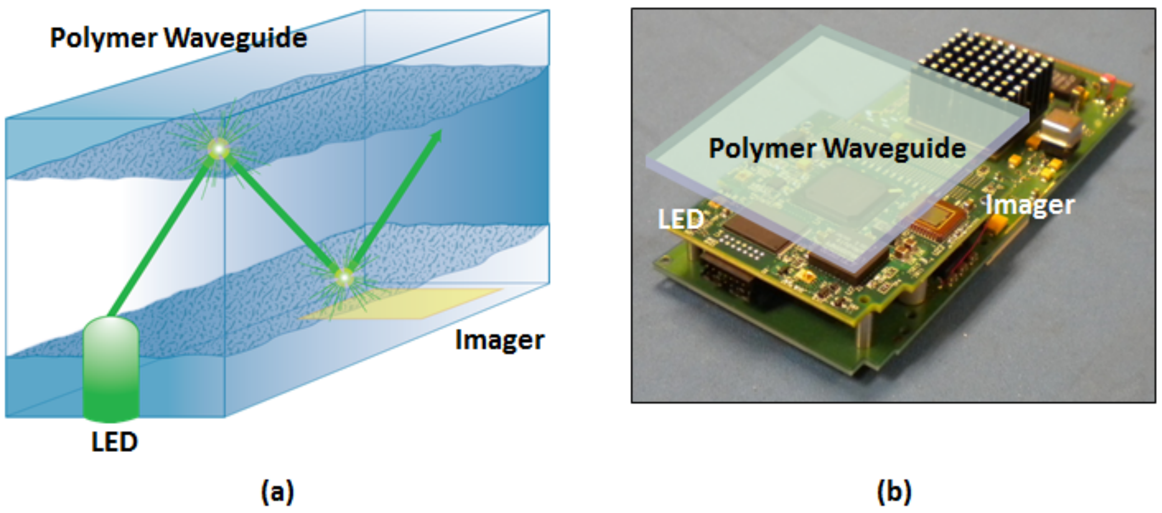


Figure 7: Optical PUF implemented with a waveguide; (a) operating concept illustration, (b) implementation on a fully fabricated PCB.

Many factors, for example, temperature, aging, etc., can cause the image to vary. A technique called fuzzy extraction is employed to ensure that the same key will be derived from the PUF under various situations [8]. This ability allows the S-COP to secure the boot process, load only trusted software, and ensure that the unique identity is intact, before, during, and after boot process. In addition to protecting data-at-rest with cryptography, the S-COP also uses key management to support secure communications between subsystems to protect data-in-transit.

This architecture allows software applications to be developed and tested initially without invoking security features. When a system is provisioned for deployment, the PUF is applied to its PCB and the finalized software code encrypted with the PUF derived key is loaded. The system will not start if its PUF value is incorrect, causing a failed software decryption. The decoupling of the S-COP and the CPU allows DoD systems to leverage mainstream CPUs, enhancing their performance, usability, and upgradability.

Figure 8 shows a testbed that we have developed to evaluate the S-COP-based secure architecture. In an unsecured architecture, the CPU reads in the basic input output system (BIOS), and bootstraps the operating system (OS). Without authentication, the CPU is vulnerable to maliciously modified BIOS and OS. The S-COP-based secure architecture addresses this vulnerability by authenticating the BIOS, OS and applications, which is illustrated in Figure 9. When the system powers up, the S-COP halts the CPU while it performs authentication. It first reads the PUF and derives a key. The key is used to decrypt the BIOS.

If successful, the CPU is released to execute the BIOS. The SCOP then authenticates and decrypts the OS, and boots the system up. Encrypted applications are loaded and handled in the same manner. In addition to associating an application with a specific system, the system can use LOCKMA dynamic key management to dynamically and seamlessly adjust the authorization of application execution (e.g., in time-specific and/or location-specific modes). Figure 10 illustrates the data protection functions of the S-COP, which uses LOCKMA to set up secure encrypted communication channels between multiple systems as well as encrypt the storage (data-at-rest).

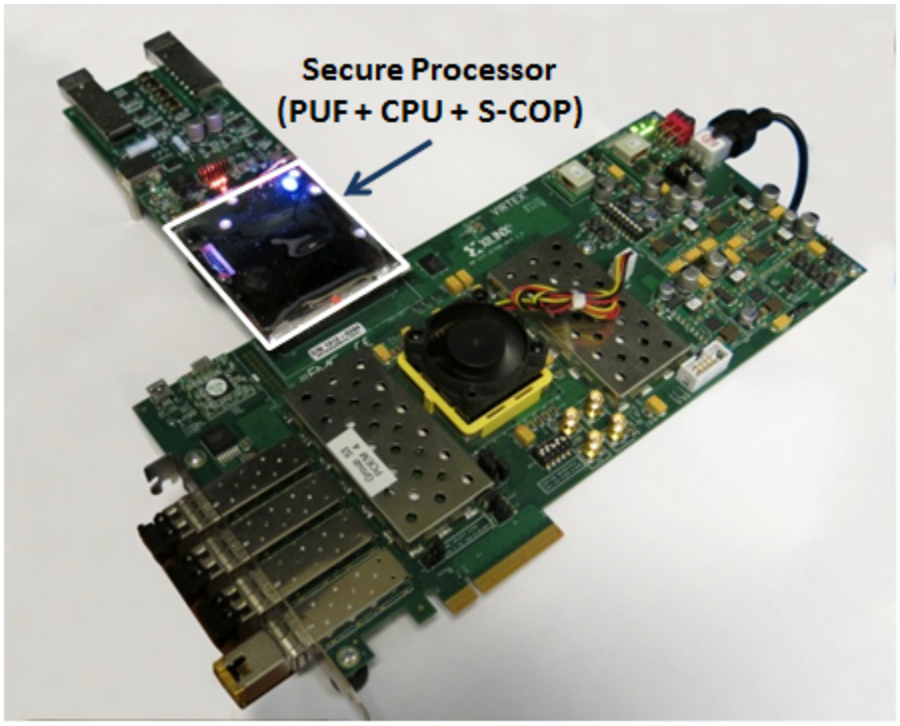


Figure 8: A secure processor integrating a CPU, an S-COP, and a PUF.

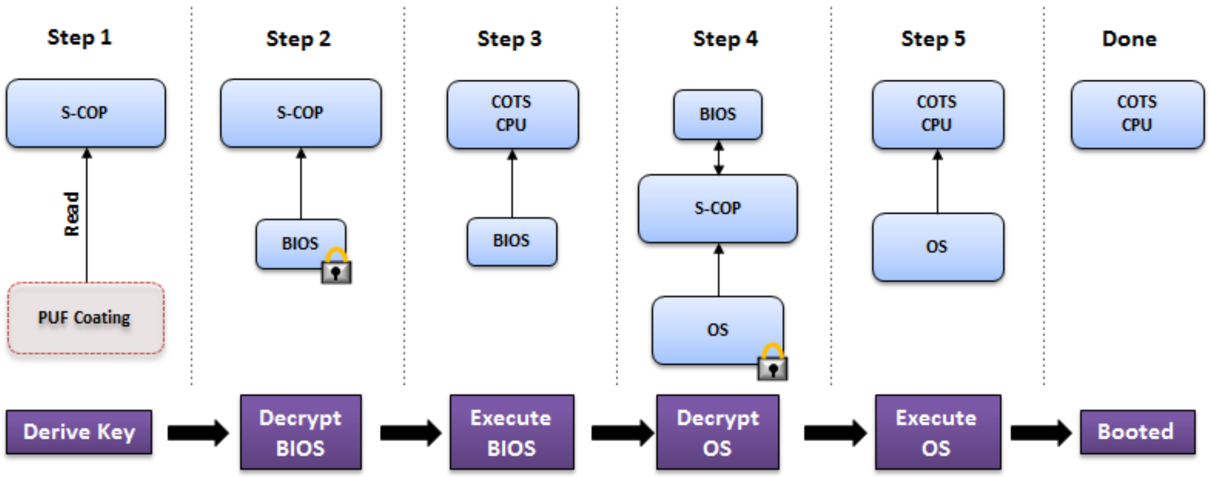
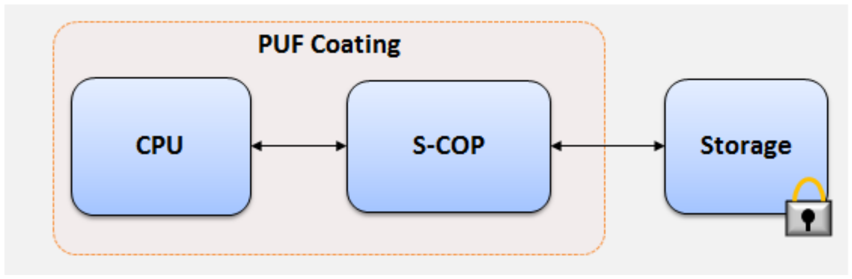
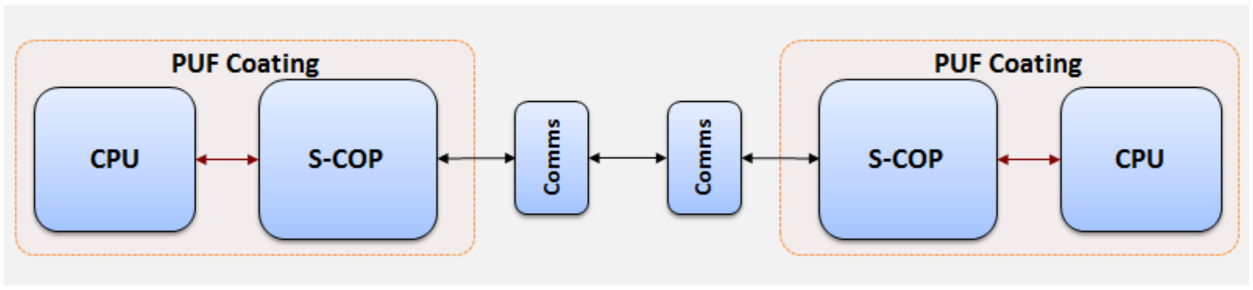


Figure 9: Secure boot process. The CPU is halted until S-COP successfully verified system integrity.



(a)



(b)

Figure 10: Data protection of (a) data-at-rest; (b) data-in-transit.

Evaluation

In terms of the CIA triad, the S-COP addresses confidentiality and integrity by protecting the boot process, data, and communication from unauthorized access and alternation. The S-COP itself does not ensure a system's availability, but it can be adapted to support other agility and resilient measures such as moving target technologies [9].

As an example, we evaluate the hypothetical UAS embedded system, after adopting the S-COP-based secure architecture, using the three security metrics: trustworthiness, protection, and usability. A mainstream unsecured CPU by itself receives low trustworthiness ratings during all system operation phases as we assume that it needs an inherently large TCB and lacks hardware enforced boot attestation. The security of such a CPU enhanced with S-COP dramatically increases across all system operation phases, earning it higher trustworthiness ratings. However, during the execution phase, one still needs to trust the operating system, which may have inherent vulnerabilities. The trusted boot does not completely eliminate the risk of running untrusted/unverified codes that could potentially escalate privileges or exfiltrate information.

If a CPU has no explicit support of volume protection, it will receive low protection ratings during the boot phase. The integration with a TPM provides key storages and measurements. However, any usage of these keys still requires the operating system to handle them properly, thus increasing the TCB. A lack of overall support for physical protection or hardware enforced encryption of code and data allows for snooping or modification of memory during the execution phase. During the off phase, the TPM could be physically replaced and thus a new set of measurements could be inserted into the system. The S-COP-based secure architecture mitigates these deficiencies by creating a root of trust with a PUF and can be used to support volume protection.

Since the S-COP can be adapted to secure a mainstream CPU, the usability of the secure UAS embedded architecture rates high as it can leverage all the benefits of a COTS CPU, such as high performance (e.g., for signal processing), large cache and memory support, and widely supported software libraries.

Open System Architecture Security

As military electronic systems continue to increase in sophistication and capability, the cost and development time of these systems also grow. The use of open system architecture (OSA) can improve the development and lifecycle efficiency of asset procurements. Typically OSA incorporates several buses with well-defined interfaces for communications (controls and data) between components. The system can then be adapted to different functionality by providing proper components and defining their interconnections.

Besides the securing of a CPU, LOCKMA is being developed into a crypto-based secure framework that has been successfully demonstrated in OSA embedded system protection. The framework employs LOCKMA, e.g., in the form of collaborating S-COPs, to provide encryption of data-in-use, data-in-transit, and data-at-rest to countermeasure eavesdropping, probing and unauthorized accessing. In addition, a trusted configuration is enforced by extending the secure boot concept and accepting only predetermined payloads and preventing unauthorized hardware and/or software substitute. An example configuration illustrated in Figure 11 consists of several payloads and processors and a LOCKMA security manager (LSM). The authorized mission configuration is specified by a digitally signed "config file" that specifies authorized payloads, allowed combinations of payloads, and secure communication channels.

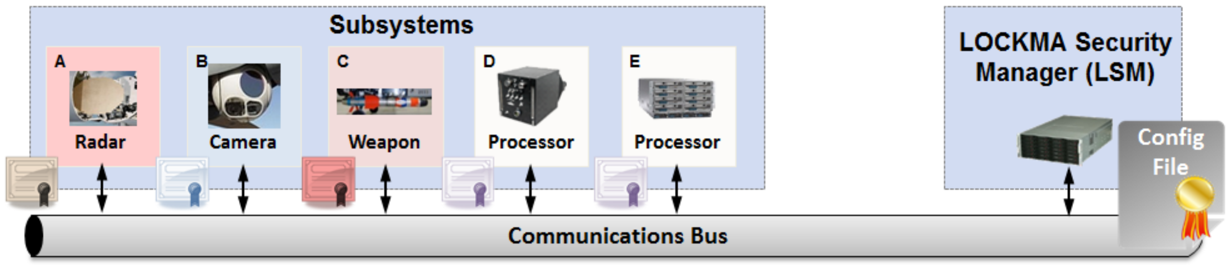
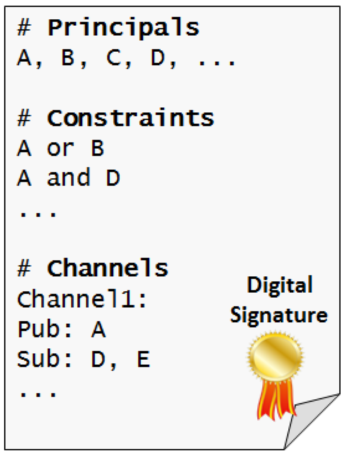


Figure 11: In a LOCKMA-based OSA security framework, LSM checks subsystem credentials against a config file to ensure that the configuration is authorized.

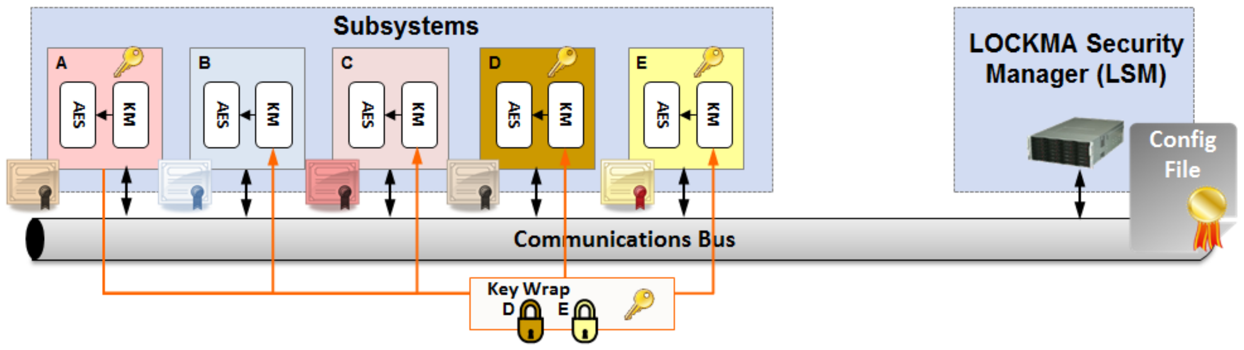


```
# Principals
A, B, C, D, ...

# Constraints
A or B
A and D
...

# Channels
Channel1:
Pub: A
Sub: D, E
...
```

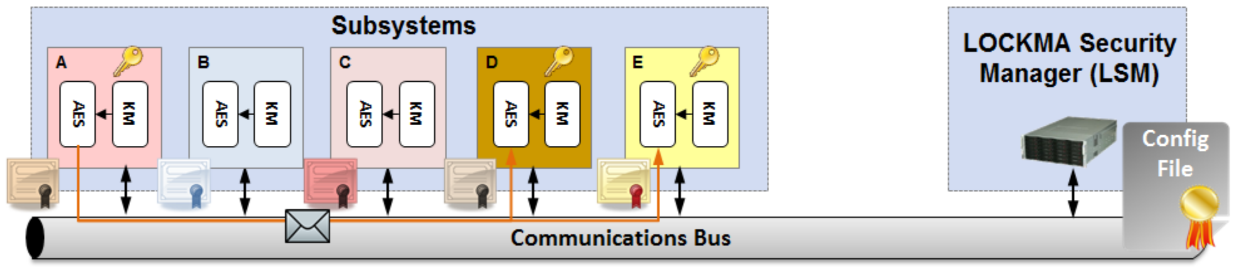
Digital Signature

Figure 12: Security configuration file for enforcing payload authorization and securing communication channels.

At start-up, the LSM verifies the digital signature of the config file and ensures that it is unaltered. Using the config file, the LSM collects subsystem credentials and confirms that the system has no unexpected payloads and its configuration (i.e., component combination) is authorized. The system now starts and the LSM continues to set up secure communication channels. Figure 13 illustrates that each subsystem is provided with, e.g., by embedding an S-COP, a key management function (KM) and an AES (advanced encryption standard) encryption/decryption function. Subsystem A creates a key wrap containing a symmetric crypto key that is only accessible by authorized subsystems D and E, and establishes a communication channel. The channel users retrieve the common secret session key and use it for encrypted communications. The system is now ready to perform its mission objectives.



(a)



(b)

Figure 13: In a LOCKMA security framework, (a) subsystem A sends a key wrap openable only by subsystems D and E to retrieve a session key, and (b) all three subsystems are then enabled to carry out encrypted communication.

Ongoing Work

Security is of asymmetric nature as an attacker can render any system defense useless by discovering a single, unexpected vulnerability. As it is impossible to correctly predict every future attack, securing an embedded system to prevent it from being attacked is not a guarantee of mission assurance. Just being secure is no longer adequate, systems must also be resilient. Lincoln Laboratory is vigorously pursuing an answer to the essential mission assurance question: “What can be done if the attacker is successful despite the best practice to provide security?”

Our objective is to define a standardized, reference security and resilient architecture for future DoD embedded systems. We want to ensure that the system continues to function when things do not go as we expected. Our work is guided by the four stages of actions involved with the resiliency of an embedded system: anticipate, withstand, recover, and evolve [10]. Our current R&D effort focuses on approaches that enable a system to withstand attacks and complete mission goals, recover from a degraded state back to a nominal state, and evolve to improve defense and resiliency against further threats.

Acknowledgements

The authors would like to acknowledge technical contributions from the team members: Branden Chetwynd, Kate Thurmer, Rob Cunningham, Kyle Ingols, M. Spain, B. Fuller, C. Walz, W. Bastow, J. Gagnon, R. Govotski, K. Gettings, F. Ennis, J. Muldavin, M. Geis, and M. Burke.

References

[1] http://www.dtic.mil/whs/directives/corres/pdf/850001_2014.pdf, accessed July 2015.

[2] D. Martinez, R. Bond, and M. Vai, ed., *High Performance Embedded Computing Handbook: A Systems Perspective*, CRC Press, 2008.

[3] DoD Open Systems Architecture Contract Guidebook for Program Managers, V. 1.1, June 2013.

[4] T. Huffmire, C Irvine, T. Nguyen, T. Levin, and R. Kastner, *Handbook of FPGA Design Security*, Springer Netherlands, 2010.

[5] "How to Use the TPM: A Guide to Hardware-Based Endpoint Security," Trusted Computing Group, 2009.

[6] "Software Guard Extensions Programming Reference," Intel, September 2013.

[7] https://www.nsa.gov/ia/programs/suiteb_cryptography/, accessed January 2015.

[8] M. Spain, B. Fuller, K. Ingols, and R. Cunningham, "Robust Keys from Physical Unclonable Functions," IEEE International Symposium on Hardware-Oriented Security and Trust, May 6, 2014.

[9] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding Focus in the Blur of Moving Target Techniques," IEEE Security & Privacy, March/April Issue, 2014.

[10] http://www.mitre.org/publications/technical-papers/cyber-resiliency-engineering-framework, accessed July 2015.